03

Enviando e recebendo mensagens JMS

Downloads

Caso queira começar o treinamento a partir dessa aula, pode baixar o projeto [aqui][1]. Baixe este arquivo, se não tiver feito os exercícios anteriormente.

Revisão do capítulo anterior

Falamos sobre o tratamento de erro. Vimos como configurar o `errorHandler`, usando um `deadLetterChannel` com `redelivery`. Mencionamos que o `deadLetterChannel` é uma adaptação de um conceito já usado na mensageria, por exemplo, JMS. Neste capítulo, vamos consumir e enviar mensagens de uma fila JMS.

Instalação do ActiveMQ

Até agora, lemos as nossas mensagens de uma pasta. Isso pode acontecer na integração (seguindo o estilo de integração *File Transfer* ou *Troca de arquivo*), mas no caso, todas as mensagens vêm de um fila JMS. Para simular a entrega de mensagens, usaremos um MOM (*Message Oriented Middleware*) muito popular no mundo Java: o ActiveMQ.

Para instalar basta baixar o ZIP da página do ActiveMQ: http://activemq.apache.org/download.html (http://activemq.apache.org/download.html)

Depois ter descompactado podemos entrar na pasta de instalação e na pasta `bin` (na linha de comando). Para subir o ActiveMQ basta rodar o script `activemq` com o parâmetro `console`.

- No Windows: `activemq.bat console`
- No Linux ou Mac: `sh activemq console`

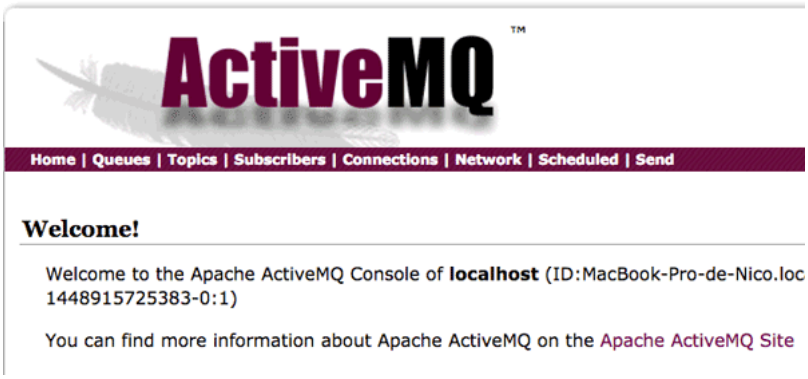
Isso faz com que ActiveMQ suba mostrando informações no console.

Obs: O Alura possui um treinamento focado no JMS com ActiveMQ que vai muito além do apresentado aqui: https://www.alura.com.br/course/jms (https://www.alura.com.br/course/jms)

Preparação da fila pedidos

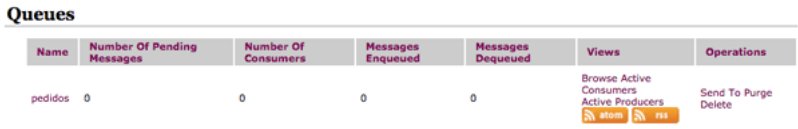
Agora já podemos acessar a interface de administração do ActiveMQ: http://localhost:8161/admin/ (http://localhost:8161/admin/)

Para os dados de Login e Senha use **admin**.

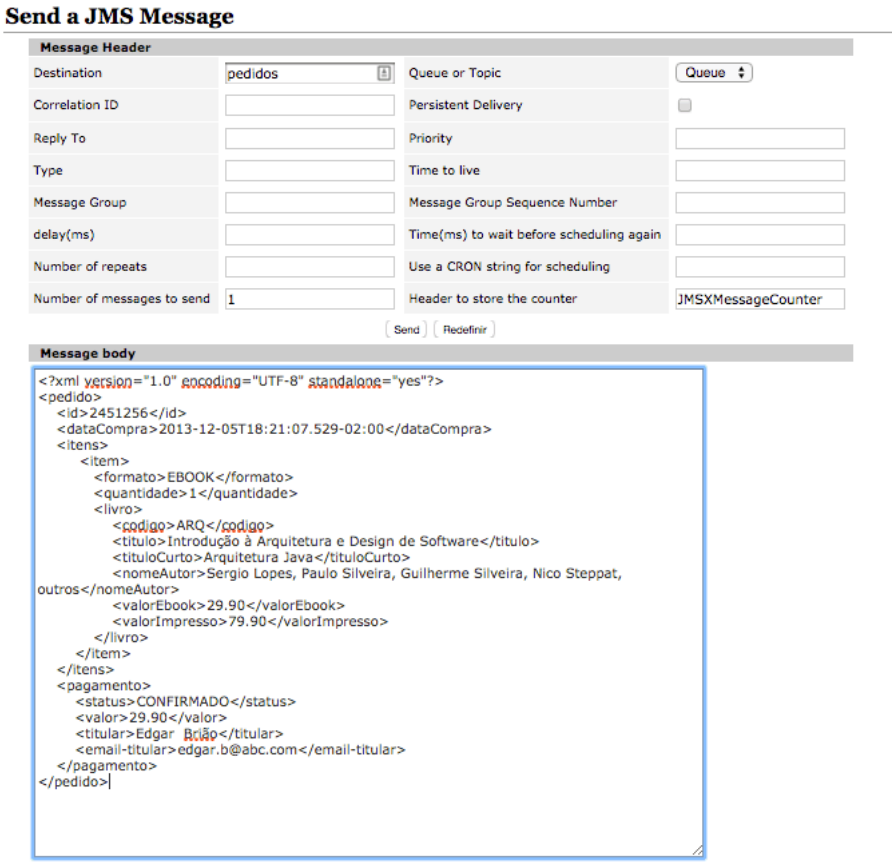


Após fazermos o login, cadastraremos uma nova fila. Clique no *Queues* e chame a fila de `pedidos` :

Uma vez criada, enviaremos mensagens pela interface web. Clique no link "Send" da coluna "Operations":



Em seguida, veremos um formulário para enviar uma mensagem JMS. No corpo da mensagem, vamos colar o XML de um pedido:



Depois, teremos uma mensagem com status *Enqueued*. Isso significa que a mensagem está pronta para ser entregue, ou pronta para ser consumida pelo Apache Camel.

Consumindo mensagens pelo Apache Camel

O Camel é integrado muito bem com JMS e ActiveMQ. Se for necessário acessar um broker JMS que não é ActiveMQ, existe um componente JMS:

http://camel.apache.org/jms.html (http://camel.apache.org/jms.html)

Como o ActiveMQ também é da Apache, foi criado um componente específico para ele:
http://camel.apache.org/activemq.html (http://camel.apache.org/activemq.html)

O primeiro passo é configurar o componente a partir do `CamelContext`. É preciso informar qual é o IP e a porta do broker, que é feito pelo método `addComponent`:

```
context.addComponent("activemq", ActiveMQComponent.activeMQComponent("tcp://localhost:61616"));
```

No segundo parâmetro adicionamos programaticamente o componente `ActiveMQComponent` ao Camel e o primeiro parâmetro será o apelido que usaremos no Camel DSL.

O componente `activemq` funciona com a mesma sintaxe dos outros componentes. Usaremos `:` e configurar se quisermos usar fila ou tópico seguido de "dois pontos" novamente para definir o nome do destino. Depois, vamos ler os pedidos da `queue` com o nome `pedidos`:

```
//usamos o componente activemq, consumindo da fila pedidos
from("activemq:queue:pedidos").
    log("${file:name}").
    routeId("rota-pedidos").
    delay(1000).
    to("validator:pedido.xsd").
    log("chegamos aqui").
    multicast().
        to("direct:soap").
            log("Chamando soap com ${body}").
        to("direct:http");
```

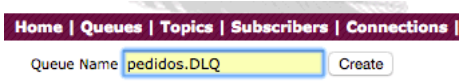
Conseguiremos consumir as mensagens! É incrível como o Camel consegue simplificar o código. Verificando a interface de administração do Apache Camel perceberemos que a mensagem foi entregue (*dequeued*):

Usando DLQ no deadLetterChannel

No último capítulo, discutimos como o Camel lida com exceções e vimos como configurar o `deadLetterChannel`. Neste `deadLetterChannel`, usaremos uma fila JMS para guardar as mensagens venenosas, algo comum na integração. O JMS tem a vantagem de guardar as mensagens de maneira persistente e segura. Então, aproveitaremos essas características para combinar o `deadLetterChannel` com uma *Dead Letter Queue* (DLQ).

O primeiro passo é criar essa fila pela interface de administração do ActiveMQ: http://localhost:8161/admin/queues.jsp (http://localhost:8161/admin/queues.jsp)

Vamos chamar essa fila de `pedidos.DLQ`:



Em seguida, ajustar o `deadLetterChannel` para enviar as mensagens venenosas:

```
errorHandler(
    deadLetterChannel("activemq:queue:pedidos.DLQ"). //usando DLQ
        logExhaustedMessageHistory(true).
        maximumRedeliveries(3).
            redeliveryDelay(5000).
        onRedelivery(new Processor() {

                @Override
                public void process(Exchange exchange) throws Exception {
                    int counter = (int) exchange.getIn().getHeader(Exchange.REDELIVERY_COUNTER);
                    int max = (int) exchange.getIn().getHeader(Exchange.REDELIVERY_MAX_COUNTER);
                    System.out.println("Redelivery - " + counter + "/" + max );
                }
        })
);
```

Vamos pegar um XML de pedido que não passe pela validação e enviar pela interface do ActiveMQ para a fila `pedidos`. Após as tentativas de entrega o Camel deve entregar a mensagem para a DLQ. Para simplificar e focar apenas no DLQ, iremos ler os arquivos XML da pasta `pedidos`. Fica a seu critério, segue o código:

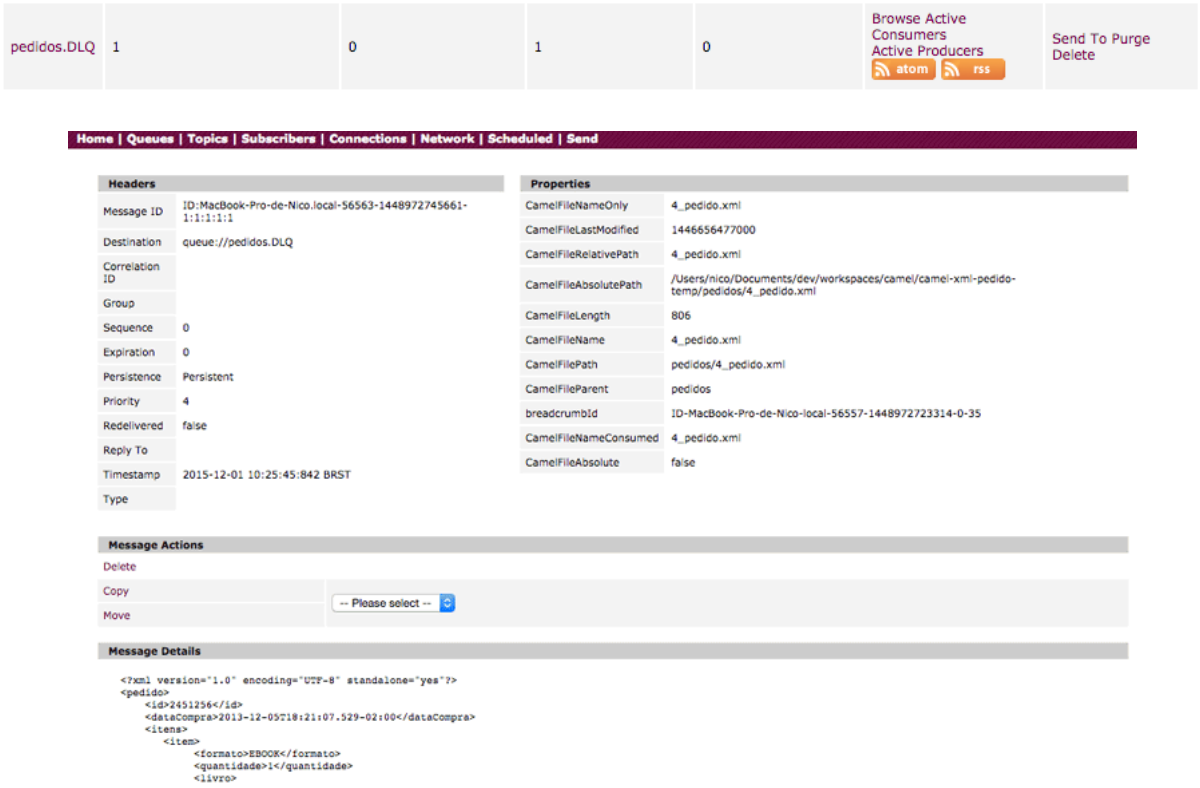
```
//from("activemq:queue:pedidos").
from("file:pedidos?delay=5s&noop=true").
    log("${file:name}").
    routeId("rota-pedidos").
//resto do codigo omitido
```

Não esqueça de rodar o Tomcat para inicializar os serviços web. Como resultado, o Camel deve entregar a XML do pedido que não passou pela validação a fila `pedidos.DLQ`.

Você pode acessar a DLQ pela interface do ActiveMQ:

http://localhost:8161/admin/browse.jsp?JMSDestination=pedidos.DLQ (http://localhost:8161/admin/browse.jsp?JMSDestination=pedidos.DLQ)

Veremos as mensagens (link *Browse* na coluna *Views*), deve aparecer o nosso XML do pedido que gerou a erro de validação:



Para saber mais: Dependências do Maven

Nesse capítulo, usamos as duas dependências do Maven:

```
<dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-camel</artifactId>
    <version>5.6.0</version>
</dependency>

<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-jms</artifactId>
    <version>${camel-version}</version>
</dependency>
```

Segue o código das rotas:

```
errorHandler(
    deadLetterChannel("activemq:queue:pedidos.DLQ").
//        logExhaustedMessageHistory(true).
        maximumRedeliveries(3).
            redeliveryDelay(5000).
        onRedelivery(new Processor() {

                @Override
                public void process(Exchange exchange) throws Exception {
                    int counter = (int) exchange.getIn().getHeader(Exchange.REDELIVERY_COUNTER);
                    int max = (int) exchange.getIn().getHeader(Exchange.REDELIVERY_MAX_COUNTER);
                    System.out.println("Redelivery - " + counter + "/" + max );
                }
```

```
        })
);

//from("activemq:queue:pedidos").
from("activemq:queue:pedidos")
    log("${file:name}").
    routeId("rota-pedidos").
    delay(1000).
    to("validator:pedido.xsd").
    multicast().
        to("direct:soap").
            //log("Chamando soap com ${body}").
        to("direct:http");

from("direct:soap").
    routeId("rota-soap").
to("xslt:pedido-para-soap.xslt").
    //log("Resultado do template: ${body}").
    setHeader(Exchange.CONTENT_TYPE,constant("text/xml")).
to("http4://localhost:8080/webservices/financeiro");

from("direct:http").
    routeId("rota-http").
        setProperty("pedidoId", xpath("/pedido/id/text()")).
        setProperty("email", xpath("/pedido/pagamento/email-titular/text()")).
    split().
        xpath("/pedido/itens/item").
    filter().
        xpath("/item/formato[text()='EBOOK']").
    setHeader("CamelFileName", simple("${file:name}.json")).
    setProperty("ebookId", xpath("/item/livro/codigo/text()")).
    setHeader(Exchange.HTTP_QUERY,
                simple("clienteId=${property.email}&pedidoId=${property.pedidoId}&ebookId=${pro
to("http4://localhost:8080/webservices/ebook/item");
}
```

O que aprendemos?

- Cadastrar um componente programaticamente;
- Consumir mensagens JMS com Camel;
- Enviar mensagens JMS;
- Usando um DLQ para o deadLetterChannel.

[1]: https://s3.amazonaws.com/caelum-online-public/camel/camel-stage-cap7.zip (https://s3.amazonaws.com/caelum-online-public/camel/camel-stage-cap7.zip)